```python
# Importing required libraries
from sqlalchemy import create_engine, Column, Float, Integer, String
from sqlalchemy.orm import sessionmaker, declarative_base
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import pathlib
import math
import warnings
import unittest
from bokeh.plotting import figure, show
from bokeh.io import output_notebook
from bokeh.models import ColumnDataSource
import subprocess
import os
'''
SECTION 1.2
'''
# Suppressing warnings
warnings.filterwarnings("ignore", category=UserWarning)

# Defining the base class for SQLAlchemy
Base = declarative_base()

# Defining the TrainingData class
class TrainingData(Base):
    __tablename__ = 'train_data'
    id = Column(Integer, primary_key=True)
    x = Column(Float)
    y1 = Column(Float)
    y2 = Column(Float)
    y3 = Column(Float)
    y4 = Column(Float)

# Defining the DataVisualization class
class DataVisualization:
    @staticmethod
    def plot_train_data_scatter():
        """
        Plot scatter plot of training data.

        This function connects to the SQLite database, retrieves training data, and plots a scatter plot
        for visualizing the relationship between 'X' and 'Y1', 'Y2', 'Y3', 'Y4'.

        Parameters:
```

```
        None

        Returns:
        None
        """
        try:
            # Connecting to SQLite database
            conn = sqlite3.connect('DataBase.db')
            query = 'SELECT * FROM train_data'
            df = pd.read_sql_query(query, conn)
            x = df['x']
            y1 = df['y1']
            y2 = df['y2']
            y3 = df['y3']
            y4 = df['y4']

            # Plotting scatter plot
            plt.title('\n "X" against Y1, Y2, Y3, Y4 from trained data \n', fontdict={'fontsize': 20, 'fontweight': 5, 'color': 'Black'})
            plt.xlabel("X values", fontdict={'fontsize': 10, 'fontweight': 10, 'color': 'blue'})
            plt.ylabel("Y Values", fontdict={'fontsize': 10, 'fontweight': 10, 'color': 'blue'})

            plt.scatter(x, y1, alpha=1, s=5, c='blue', label='X vs Y1')
            plt.scatter(x, y2, alpha=1, s=5, c='red', label='X vs Y2')
            plt.scatter(x, y3, alpha=1, s=5, c='green', label='X vs Y3')
            plt.scatter(x, y4, alpha=1, s=5, c='yellow', label='X vs Y4')

            plt.legend()
            plt.show()

            # Closing database connection
            conn.close()
        except sqlite3.Error as e:
            print(f"SQLite error: {e}")
        except Exception as e:
            print(f"An error occurred: {e}")

# Defining the DataProcessing class
class DataProcessing:
    def find_sum_of_least_squares(self, x, y):
        """
        Find the sum of least squares between two arrays.

        Parameters:
        x (numpy.array): First array.
        y (numpy.array): Second array.

        Returns:
        float: Sum of least squares.
```

```python
        """
        try:
            x = np.array(x)
            y = np.array(y)
            difference = np.subtract(x, y)
            square = np.square(difference)
            sum_of_squares = np.sum(square)
            return sum_of_squares
        except Exception as e:
            print(f"An error occurred: {e}")

    def find_least_squares(self, x, y):
        """
        Find the least squares between two arrays.

        Parameters:
        x (numpy.array): First array.
        y (numpy.array): Second array.

        Returns:
        pandas.DataFrame: DataFrame containing least squares values.
        """
        try:
            x = np.array(x)
            y = np.array(y)
            difference = np.subtract(x, y)
            square = np.square(difference)
            return pd.DataFrame(square)
        except Exception as e:
            print(f"An error occurred: {e}")

    def find_deviation(self, x, y):
        """
        Find the deviation between two arrays.

        Parameters:
        x (numpy.array): First array.
        y (numpy.array): Second array.

        Returns:
        pandas.DataFrame: DataFrame containing deviation values.
        """
        try:
            x = np.array(x)
            y = np.array(y)
            difference = np.subtract(x, y)
            return pd.DataFrame(difference)
        except Exception as e:
            print(f"An error occurred: {e}")

    def any_deviation_greater_than_threshold(self, x, y, threshold):
        """
        Check if any deviation is greater than the threshold.

        Parameters:
        x (numpy.array): First array.
        y (numpy.array): Second array.
        threshold (float): Threshold value.

        Returns:
        bool: True if any deviation is greater than the threshold, False otherwise.
        """
        try:
            x = np.array(x)
            y = np.array(y)
            difference = pd.DataFrame(np.subtract(x, y))
            return (difference > threshold).any().any()
        except Exception as e:
            print(f"An error occurred: {e}")

# Defining the DataLoader class
class DataLoader:
    def __init__(self, data_base, table_name, data_frame):
        self.Base = Base
        self.data_base = data_base
        self.table_name = table_name
        self.data_frame = data_frame
        self.engine = create_engine('sqlite:///{}.db'.format(self.data_base))

    def load_data(self):
        """
        Load data into the SQLite database.

        Parameters:
        None

        Returns:
        None
        """
        try:
            self.Base.metadata.create_all(self.engine)
            table_name = self.table_name
            self.data_frame.to_sql(table_name, con=self.engine, if_exists='replace', index=False)
        except Exception as e:
            print(f"An error occurred: {e}")

    def close_connection(self):
        """
        Close the SQLite database connection.

        Parameters:
        None

        Returns:
        None
        """
        try:
            self.engine.dispose()
        except Exception as e:
            print(f"An error occurred: {e}")

# Defining the Ideal class
class Ideal(DataProcessing):
    def load_ideal_data(self):
        """
        Load ideal data from the CSV file.

        Parameters:
        None

        Returns:
        pandas.DataFrame: Loaded ideal data.
        """
        try:
            self.ideal = pd.read_csv("Datasets/ideal.csv")
            return self.ideal
        except FileNotFoundError as e:
            print(f"File not found: {e}")
        except Exception as e:
            print(f"An error occurred: {e}")

    def find_ideal(self, x):
        """
        Find the ideal functions based on least squares.

        Parameters:
        x (numpy.array): Array for which ideal functions are calculated.

        Returns:
        pandas.DataFrame: DataFrame containing ideal functions.
        """
        try:
            database = DataProcessing()
            least_squares = [database.find_sum_of_least_squares(x, self.ideal.iloc[:, i]) for i in range(len(self.ideal.columns))]
            first_four_least_squares = sorted(least_squares)[:4]
            indices = [least_squares.index(i) for i in first_four_least_squares]
            ideal_functions = [self.ideal.iloc[:, i] for i in indices]
            ideal_functions = pd.DataFrame(ideal_functions).transpose()
            ideal_functions.columns = ['y1', 'y2', 'y3', 'y4']
            return ideal_functions
        except Exception as e:
            print(f"An error occurred: {e}")

# Defining the Test class
class Test(DataProcessing):
    def load_test_data(self):
        """
        Load test data from the CSV file.

        Parameters:
        None

        Returns:
        pandas.DataFrame: Loaded test data.
        """
        try:
            self.test = pd.read_csv("Datasets/test.csv")
            return self.test
        except FileNotFoundError as e:
            print(f"File not found: {e}")
        except Exception as e:
            print(f"An error occurred: {e}")

# Defining the Train class
class Train(DataProcessing):
    def load_training_data(self):
        """
        Load training data from the CSV file.

        Parameters:
        None

        Returns:
        pandas.DataFrame: Loaded training data.
        """
        try:
            self.train = pd.read_csv("Datasets/train.csv")
            return self.train
        except FileNotFoundError as e:
            print(f"File not found: {e}")
        except Exception as e:
            print(f"An error occurred: {e}")

    def get_deviation(self, x):
        """
        Get deviation between x and y values in the training data.

        Parameters:
        x (numpy.array): Array for which deviation is calculated.

        Returns:
        pandas.DataFrame: DataFrame containing deviation values.
        """
        try:
            self.deviation = self.find_deviation(x, self.train.iloc[:, 1])
            return self.deviation
        except Exception as e:
            print(f"An error occurred: {e}")

# Defining the TestYourCode class
class TestYourCode(unittest.TestCase):
    def test_find_sum_of_least_squares(self):
        data_processing = DataProcessing()
        x = [1, 2, 3, 4, 5]
        y = [2, 4, 6, 8, 10]
        result = data_processing.find_sum_of_least_squares(x, y)
        self.assertEqual(result, 55)

    def test_find_least_squares(self):
        data_processing = DataProcessing()
        x = [1, 2, 3, 4, 5]
        y = [2, 4, 6, 8, 10]
        result = data_processing.find_least_squares(x, y)
        expected_result = pd.DataFrame([1, 4, 9, 16, 25], columns=['x'])
        pd.testing.assert_frame_equal(result, expected_result)

    def test_find_deviation(self):
        data_processing = DataProcessing()
        x = [1, 2, 3, 4, 5]
        y = [2, 4, 6, 8, 10]
        result = data_processing.find_deviation(x, y)
        expected_result = pd.DataFrame([-1, -2, -3, -4, -5], columns=['x'])
        pd.testing.assert_frame_equal(result, expected_result)

    def test_any_deviation_greater_than_threshold(self):
        data_processing = DataProcessing()
        x = [1, 2, 3, 4, 5]
        y = [2, 4, 6, 8, 10]
        threshold = 5
        result = data_processing.any_deviation_greater_than_threshold(x, y, threshold)
        self.assertTrue(result)

# Main block of code
if __name__ == "__main__":
    try:
        # Defining file paths and names
        path = pathlib.Path(__file__).parent.resolve()
        dataset_path = os.path.join(path, "Datasets")
        ideal_filename = "ideal"
        train_filename = "train"
        database_name = "DataBase"

        # Reading and loading ideal data
        df_ideal = pd.read_csv(os.path.join(dataset_path, "{}.csv".format(ideal_filename)))
        ideal_table_name = ideal_filename + "_data"
        ideal_data_loader = DataLoader(database_name, ideal_table_name, df_ideal)
        ideal_data_loader.load_data()
        ideal_data_loader.close_connection()

        # Reading and loading training data
        df_train = pd.read_csv(os.path.join(dataset_path, "{}.csv".format(train_filename)))
        train_table_name = train_filename + "_data"
        train_data_loader = DataLoader(database_name, train_table_name, df_train)
        train_data_loader.load_data()
        train_data_loader.close_connection()

        # Creating instances of classes
        train = Train()
        test = Test()
        ideal = Ideal()
        data_processing = DataProcessing()

        # Loading data
        train_data = train.load_training_data()
        test_data = test.load_test_data()
        ideal_data = ideal.load_ideal_data()

        # Finding ideal functions
        ideal_functions = ideal.find_ideal(train_data.iloc[:, 1])
        ideal_functions.insert(0, 'x', train_data.iloc[:, 0])

        # Calculating deviations
        deviation_between_training_and_ideal = pd.DataFrame([])
        for column in ideal_functions.columns:
            deviation_between_training_and_ideal[column] = data_processing.find_deviation(train_data.iloc[:, 1], ideal_functions[column])
        deviation_between_training_and_ideal = pd.DataFrame(deviation_between_training_and_ideal)

        # Calculating absolute deviation
        absolute_deviation = deviation_between_training_and_ideal.abs()
        maximum_deviation = absolute_deviation.max().max()

        # Calculating sqrt(2) * maximum deviation
        sqrt_2 = math.sqrt(2)
        sqrt_2_maximum_deviation = sqrt_2 * maximum_deviation

        # Initializing arrays for x and y values
        x_values = np.array([])
        y1_values = np.array([])
        y2_values = np.array([])
        y3_values = np.array([])
        y4_values = np.array([])

        # Finding best fit values
        for t in test_data.iloc[:, 0]:
            least_squares = np.array([(x - t) ** 2 for x in ideal_functions.iloc[:, 0]])

            index = np.argmin(least_squares)
            x_values = np.append(x_values, ideal_functions.iloc[index, 0])
            y1_values = np.append(y1_values, ideal_functions.iloc[index, 1])
            y2_values = np.append(y2_values, ideal_functions.iloc[index, 2])
            y3_values = np.append(y3_values, ideal_functions.iloc[index, 3])
            y4_values = np.append(y4_values, ideal_functions.iloc[index, 4])

        # Creating DataFrame for best fit values
        best_fit_values = pd.DataFrame([x_values, y1_values, y2_values, y3_values, y4_values]).transpose()
        best_fit_values.columns = ['x', 'y1', 'y2', 'y3', 'y4']

         # Creating DataVisualization object
        data_visualization = DataVisualization()

        # Initializing lists for DataFrames
        y1 = pd.DataFrame([])
        y2 = pd.DataFrame([])
        y3 = pd.DataFrame([])
        y4 = pd.DataFrame([])

        table_list = [y1, y2, y3, y4]

        # Creating DataFrames for each ideal function
        for i in range(1, 5):
            if data_processing.any_deviation_greater_than_threshold(
                    best_fit_values.iloc[:, i], test_data.iloc[:, 1], sqrt_2_maximum_deviation):
                print('y' + str(i) + ' is not in range')
            else:
                table_list[i - 1]['x'] = test_data.iloc[:, 0]
                table_list[i - 1]['y'] = test_data.iloc[:, 1]
                table_list[i - 1]['delta'] = data_processing.find_deviation(test_data.iloc[:, 1],
                                                                            best_fit_values.iloc[:, i])
                table_list[i - 1]['ideal function'] = best_fit_values.iloc[:, i]

                # Loading test data into SQLite database
                test_loader = DataLoader(database_name, 'test_' + 'y' + str(i + 1), table_list[i - 1])
                test_loader.load_data()

        # Loading training data and ideal functions into SQLite database
        training_loader = DataLoader(database_name, 'training_data', best_fit_values)
        training_loader.load_data()
        training_loader.close_connection()

        ideal_loader = DataLoader(database_name, 'ideal_functions', ideal_data)
        ideal_loader.load_data()
        ideal_loader.close_connection()

        # Plotting the scatter plot
        data_visualization.plot_train_data_scatter()

    except Exception as e:
        print(f"An error occurred: {e}")



'''
SECTION 1.3
'''
# Set your GitHub username and repository name
github_username = "Name22" #please enter your github username
repository_name = "test"#please enter your repo name in github
repository_url = f"https://github.com/{github_username}/{repository_name}.git"

# Set the directory where you want to initialize the repository
repository_directory = "C:\\Users\\Name\\Downloads\\Python Assignment-2\\Python Assignment\\Python Assignment\\Task 2 Assignment" #Refer to this to amend the directory on your local machine

# Change to the repository directory
os.chdir(repository_directory)

# Initialize a new Git repository
subprocess.run(["git", "init"])

# Add all files to the repository
subprocess.run(["git", "add", "."])

# Commit the changes
subprocess.run(["git", "commit", "-m", "Initial commit"])

# Add the GitHub remote repository
subprocess.run(["git", "remote", "add", "origin", repository_url])

```

```python
# Push the changes to the 'develop' branch
result_push = subprocess.run(["git", "push", "-u", "origin", "develop"])

# Print the result of pushing changes
print("Result of pushing changes:", result_push)

# Print a message indicating success
print("Changes pushed successfully.")


```